Understanding the Performance of Hybrid (distributed/shared memory) Programming Model

http://www.mcs.anl.gov/petsc-fun3d

William Gropp, Argonne National Laboratory

Dinesh Kaushik, Argonne National Laboratory & ODU

David Keyes, Old Dominion University, LLNL & ICASE

Barry Smith, Argonne National Laboratory

Organization of the Presentation

- Background of Message Passing (MPI)
- Background of Shared Memory Model (OpenMP)
- Bottlenecks to Parallel Scalability
- Adapting to the hybrid (MPI/OpenMP) programming model
- Conclusions and Future Work

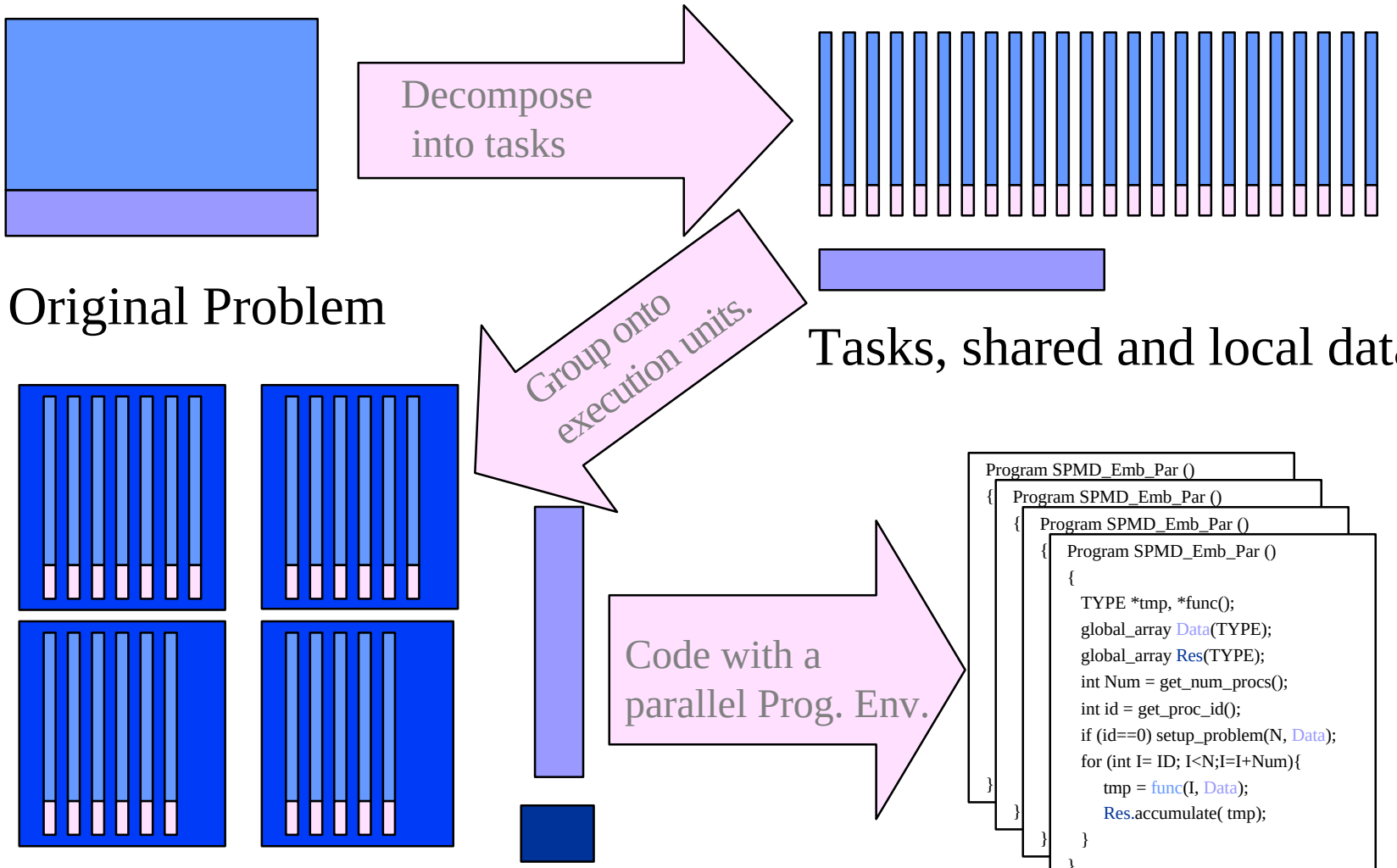
Parallel Computing

What is it?

- Parallel computing is the use of concurrency in program either to:
  - **decrease the runtime** for the solution to a problem.
  - **Increase the size** of the problem that can be solved.

Parallel Computing gives you more performance to throw at your problems.

Parallel Computing:
Writing a parallel application.



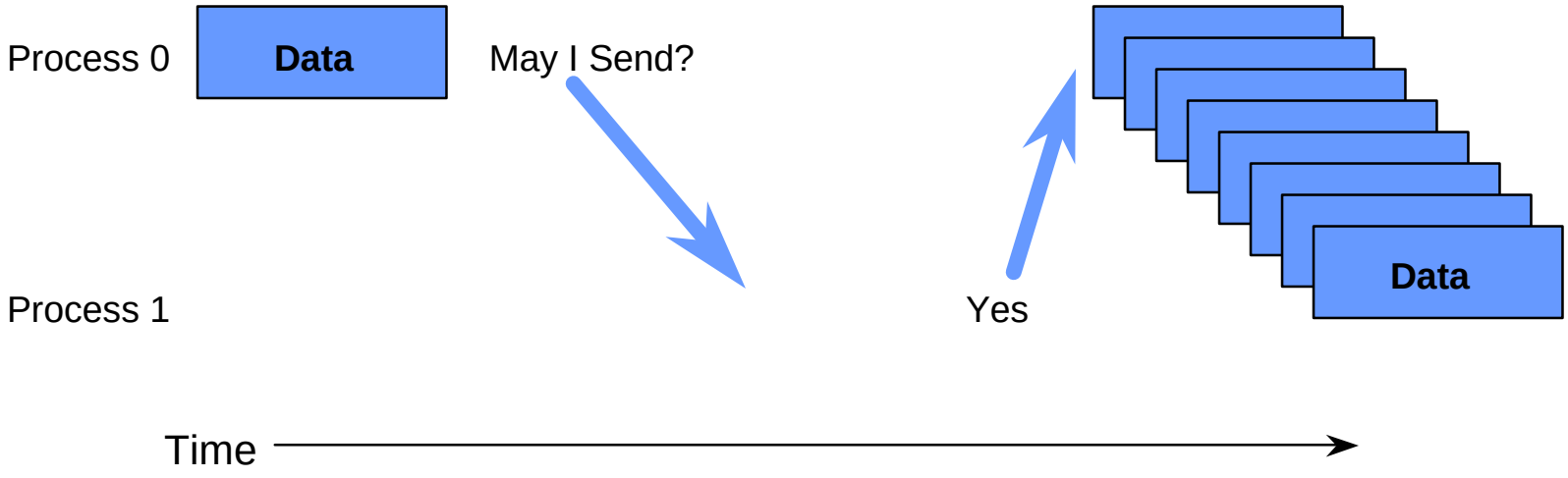
Types of Parallel Computing Models

- Data Parallel - the same instructions are carried out simultaneously on multiple data items (SIMD)
- Task Parallel - different instructions on different data (MIMD)
- SPMD (single program, multiple data) not synchronized at individual operation level
- SPMD is equivalent to MIMD since each MIMD program can be made SPMD (similarly for SIMD, but not in practical sense.)

Message passing (and MPI) is for MIMD/SPMD parallelism. HPF is an example of an SIMD interface.

What is message passing?

- Data transfer plus synchronization



- Requires cooperation of sender and receiver

What is MPI?

- A *message-passing library specification*
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Full-featured
- Designed to provide access to advanced parallel hardware for
  - end users
  - library writers
  - tool developers

Why Use MPI?

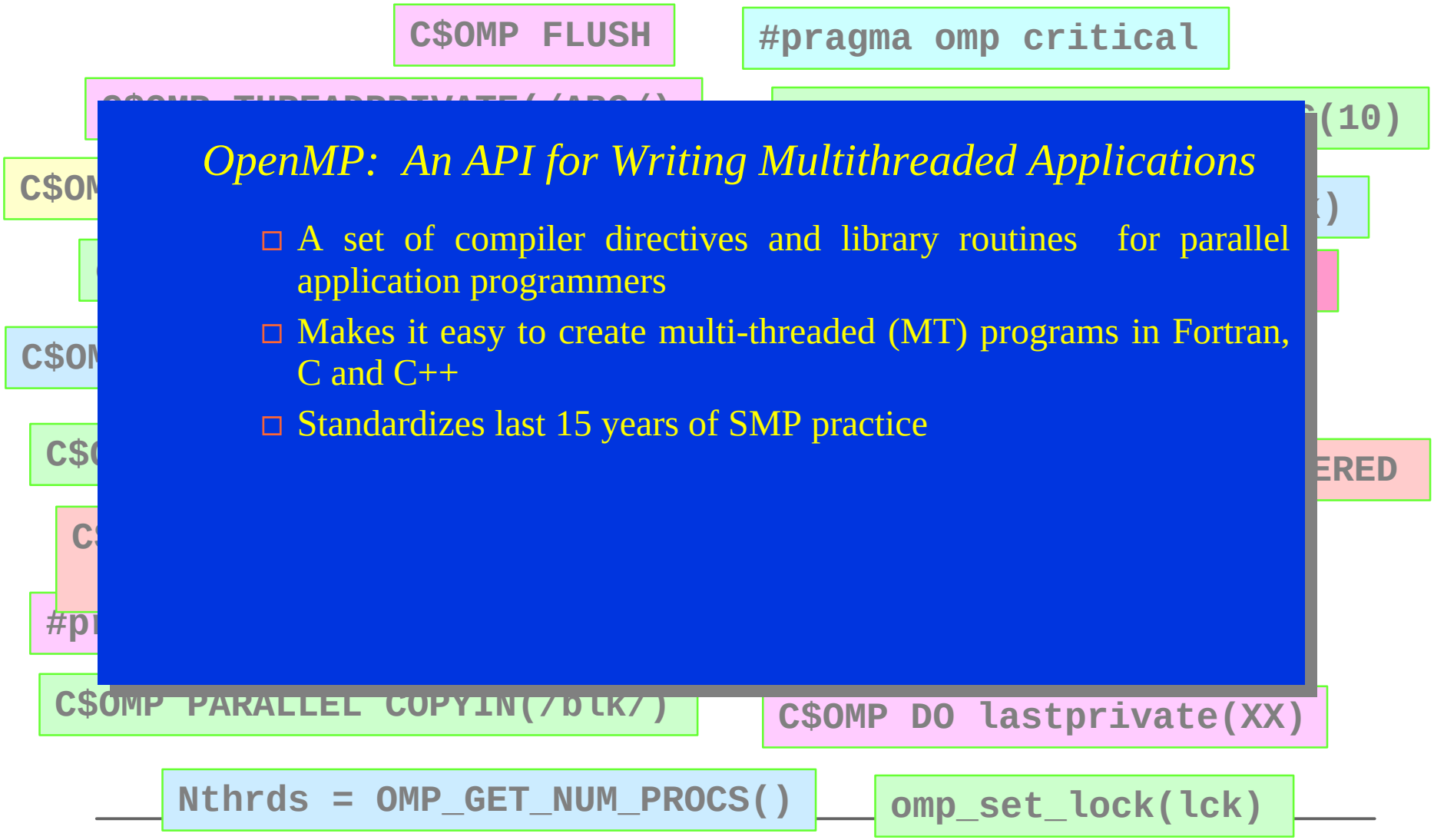
- MPI provides a powerful, efficient, and *portable* way to express parallel programs
- MPI was explicitly designed to enable libraries…
- … which may eliminate the need for many users to learn (much of) MPI

A Minimal MPI Program (C)

```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

OpenMP: Introduction

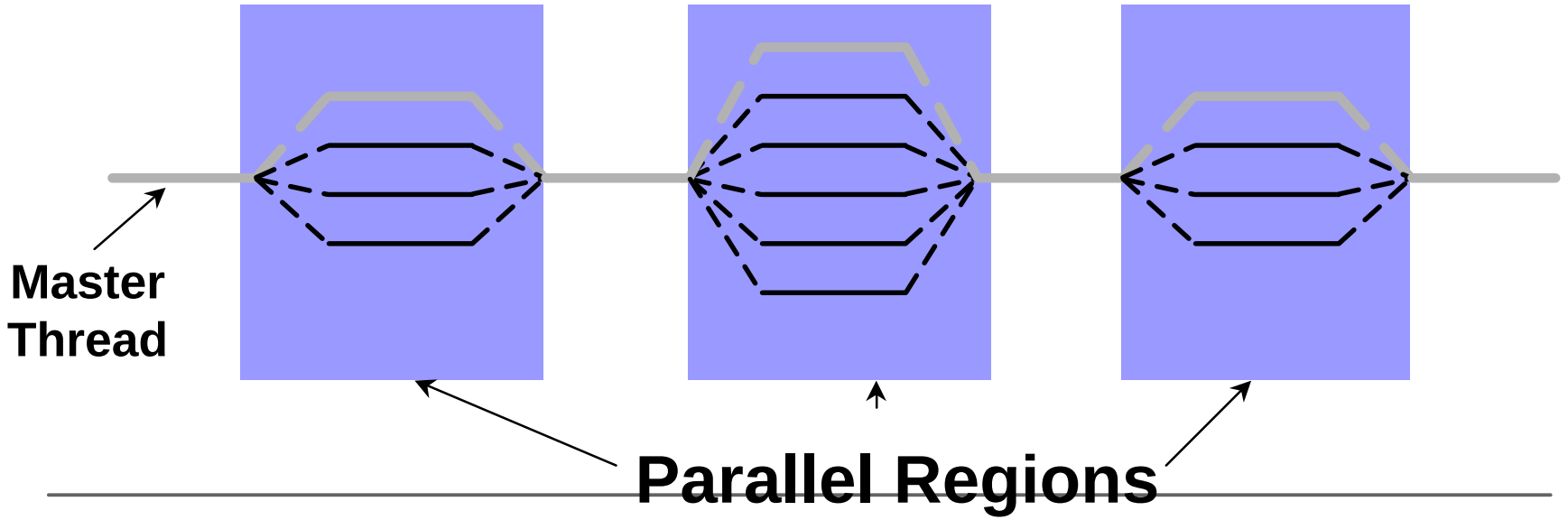


OpenMP: An API for Writing Multithreaded Applications

- A set of compiler directives and library routines for parallel application programmers
- Makes it easy to create multi-threaded (MT) programs in Fortran, C and C++
- Standardizes last 15 years of SMP practice

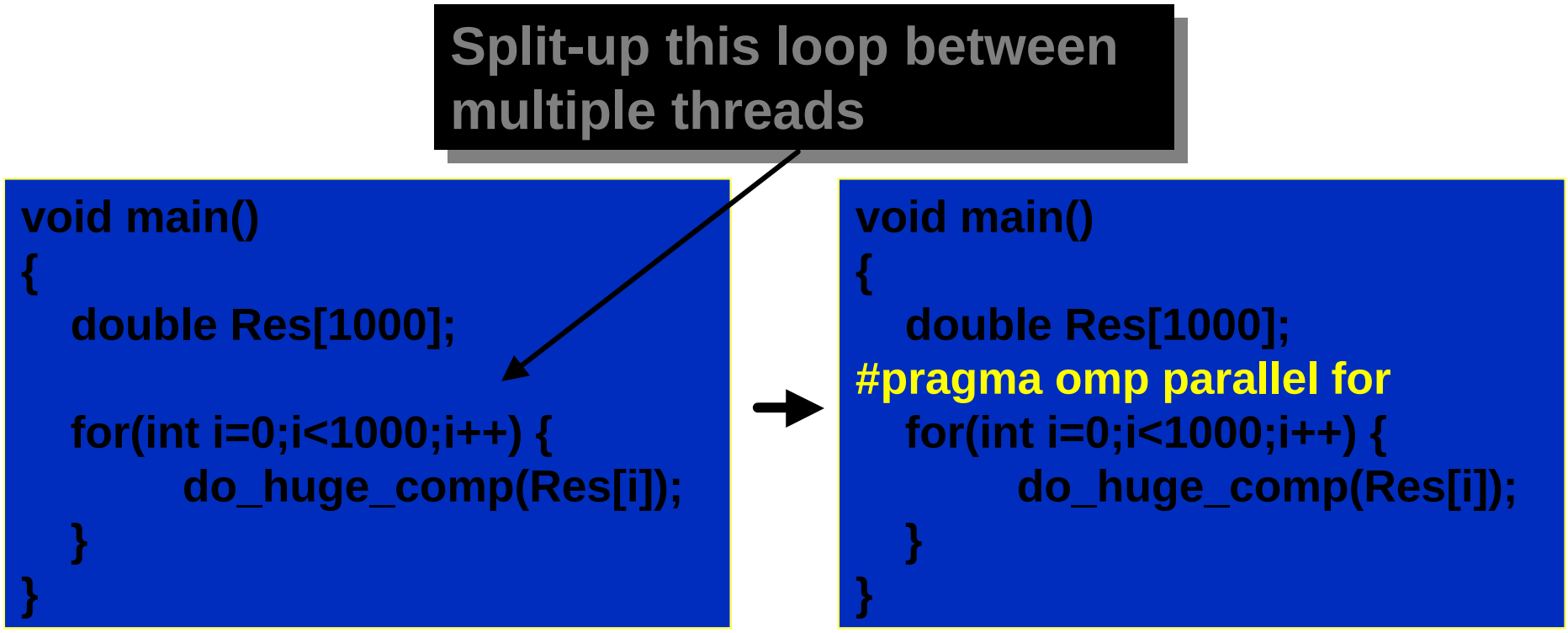
OpenMP: Programming Model

- Master thread spawns a team of threads as needed.
- Parallelism is added incrementally: i.e. the sequential program evolves into a parallel program.



OpenMP: How OpenMP is typically used?

- OpenMP is usually used to parallelize loops:
  - Find your most time consuming loops.
  - Split them up between threads.



OpenMP: The Good

- Effective Incremental Parallelism
  - Important contributor to ASCI Red results (not exactly OpenMP, but same philosophy)
- Good SMP and SMP-cluster match
  - Larger domain decomposition blocks
  - Dynamic load balance

OpenMP: The Not so Good

- Performance
  - In apples-to-apples comparison with MPI
  - Data placement important
  - Cache blocking etc. mismatch with OpenMP loop scheduling
- Restrictions on atomic update/reduce
  - No vector reduce (p 29) (but see OpenMP 2.0)
  - Complexity *for user* comes from exceptions and limitations

OpenMP: The Bad

- Program correctness
  - It is too easy to write incorrect programs
- Software Modularity
  - At best 2-level modularity
  - Many modern algorithms built out of components; how will OpenMP support them?
  - E.g., each component uses limited parallelism to fit problem into local caches; application uses task parallelism to perform intelligent (not exhaustive parameter-space search) design optimization.

Software Modularity

- Libraries must either
  - Use OpenMP at "leaves" (e.g., the loop-level), or
  - Take complete control (user program has no OpenMP parallelism when library is called).
  - But some libraries call other library routines …
    - E.g., should BLAS use OpenMP? LAPACK? What if user uses OpenMP for task parallelism for a routine that calls an LAPACK routine?
- Using OpenMP at loop-level incurs startup costs
  - Some vendors suggest
    - Program Main
      !omp parallel
      …
      !omp end parallel
      stop
      end
- OpenMP language bindings poorly chosen for mixed-language programming
  - I.e., programs that use libraries …

Language Bindings for Mixed Language Programming

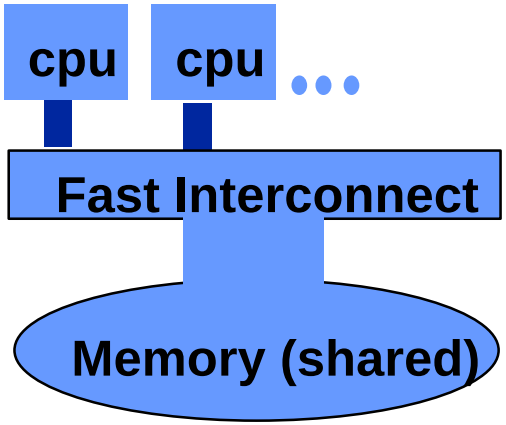
- Libraries used by Fortran may be written in C (and vice versa)
  - OpenMP naming convention can make this (nearly) impossible
- C names should *always* be distinguishable from Fortran names
  - Unless bindings are *identical*
  - Using mixed case for C (as in MPI) is an easy way to do this

Performance Issues for OpenMP

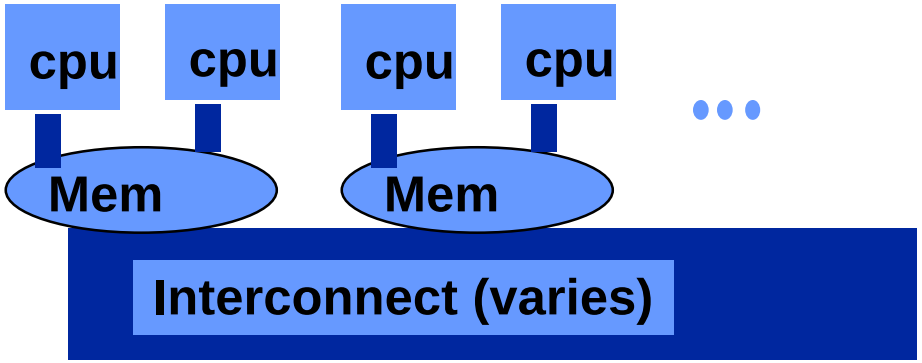
- Overhead of thread management
- Redundant storage and work
- Sequential reduction phase, which tend to be memory bandwidth bound
- Simplicity goes away when user takes care of memory updates (similar to MPI model)

Hardware Architecture

SMP Symmetric Multi-Processor



Cluster of SMP



- **2-128 processors circa late 90's**
- **Memory shared**
- **High-powered Processors and Micros (some vector, mostly micro)**
- **SMP means equal access including I/O**
- **Sometimes term is generalized to mean Shared Memory Parallel**

40 - 4000 circa 90's
Memory physically distributed
High-Powered Micros (Alpha, PowerPC)

Motivation for Hybrid Model

- Given
  - a scalable MPI based code
- Goal
  - use hybrid model to achieve better performance than MPI alone
- Methodology:
  - assign one subdomain to one MPI process
  - use OpenMP with in a subdomain that gets mapped to a node (with 2 or more processors)
- Advantage
  - take advantage of shared memory programming within a subdomain
  - results in bigger subdomains as more than one thread can work on a subdomain as compared to pure MPI case

Our View of the Hybrid Model

- MPI Extreme
  - the user manages the memory updates
- OpenMP Extreme
  - the system manages the memory updates
- Hybrid MPI/OpenMP
  - Some memory updates are managed by the user and the rest by the system

Competing for the Available Memory Bandwidth

- The processors on a node compete for the available memory bandwidth
- The computational phases that are memory-bandwidth limited will not scale
  - They may even run slower because of the extra synchronizations

Stream Benchmark on ASCI Red
MB/s for the Triad Operation

Vector Size	1 Thread	2 Threads
1E04	666	1296
5E04	137	238
1E05	140	144
1E06	145	141
1E07	157	152

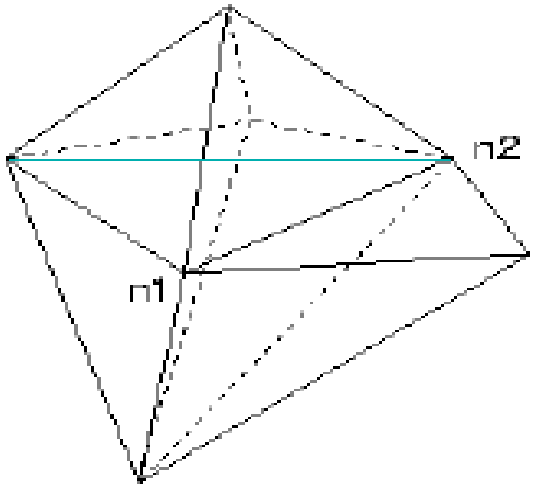
Primary PDE Solution Kernels

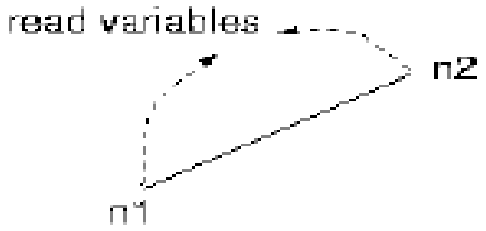
- **Vertex-based loops**
  - state vector and auxiliary vector updates
- **Edge-based "stencil op" loops**
  - residual evaluation
  - approximate Jacobian evaluation
  - Jacobian-vector product (often replaced with matrix-free form, involving residual evaluation)
- **Sparse, narrow-band recurrences**
  - approximate factorization and back substitution
- **Vector inner products and norms**
  - orthogonalization/conjugation
  - convergence progress and stability checks

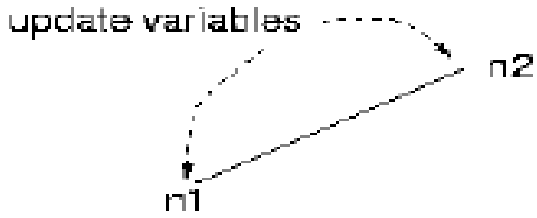
Features of PETSc-FUN3D

- Based on “legacy” (but contemporary) CFD application with significant F77 code reuse
- Portable, message-passing library-based parallelization, run on NT boxes through Tflop/s ASCI platforms
- Simple multithreaded extension (for SMP Clusters)
- Sparse, unstructured data, implying memory indirection with only modest reuse
- Wide applicability to other implicitly discretized multiple-scale PDE workloads - of interagency, interdisciplinary interest
- Extensive profiling has led to follow-on algorithmic research

Flux Evaluation in PETSc-FUN3D







Variables at each node:
- density,
- momentum (x,y,z),
- energy,
- pressure

Variables at edge:
- identity of nodes,
- orientation(x,y,z)

Key Features of Implementation Strategy

- Follow the “owner computes” rule under the dual constraints of minimizing the number of messages and overlapping communication with computation
- Each processor “ghosts” its stencil dependences in its neighbors
- Ghost nodes ordered after contiguous owned nodes
- Domain mapped from (user) global ordering into local orderings
- Scatter/gather operations created between **local sequential** vectors and **global distributed** vectors, based on runtime connectivity patterns
- Newton-Krylov-Schwarz operations translated into local tasks and communication tasks
- Profiling used to help eliminate performance bugs in communication and memory hierarchy

Factoring out the Parallel Performance

- Implementation Scalability
  - Problem constrained scalability
  - Memory constrained scalability
- Algorithmic Scalability
  - Degrades as the number of processors increase
- Per-processor Performance
  - Needs attention to the memory hierarchy

MPI: Parallel Performance on ASCI Red

ONERA M6 Wing Test Case, Tetrahedral grid of 2.8 million vertices (about 11 million unknowns) on up to 3072 ASCI Red Nodes (each with dual Pentium Pro 333 MHz processors)

Nodes	Iter	Time in seconds	Speedup	Parallel Efficiency		
				Overall	Algorithmic	Implementation
128	22	2,039	1.00	1.00	1.00	1.00
512	26	638	3.20	0.80	0.85	0.94
1024	29	362	5.63	0.70	0.76	0.93
2048	32	208	9.78	0.61	0.69	0.89
3072	34	159	12.81	0.53	0.65	0.82

MPI: Scalability Bottlenecks on ASCI Red

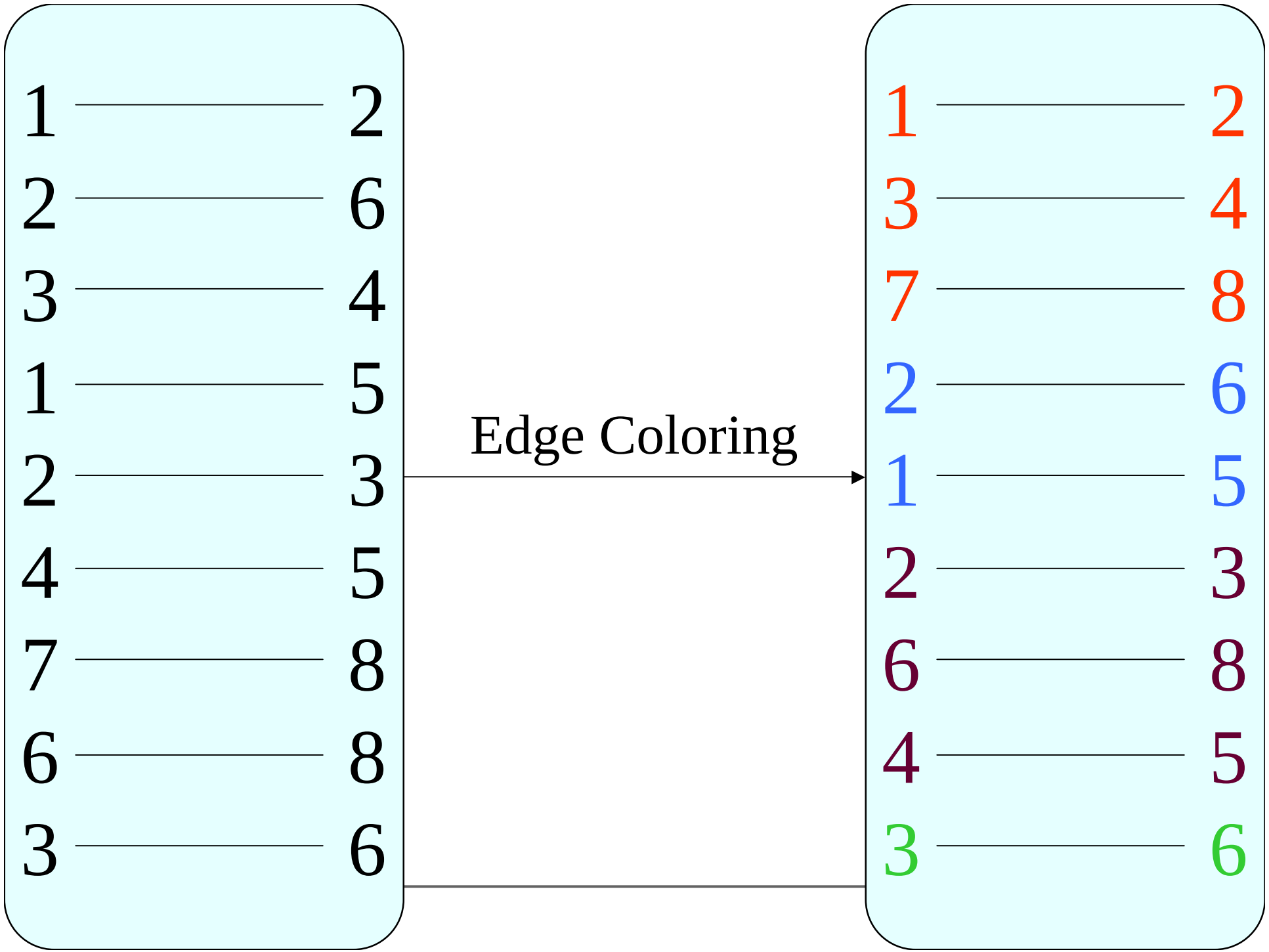
Nodes	Percentage Times for			Scatter Scalability	
	Global Reduc-tions	Implicit Synchro-nizations	Ghost Point Scatters	Total Data Sent per Iteration (GB)	Application Level Effective Bandwidth per Node (MB/s)
128	5	4	3	3.6	6.9
512	3	7	5	7.1	6.0
1024	3	10	6	9.4	7.5
2048	3	11	8	11.7	5.7
3072	5	14	10	14.2	4.6

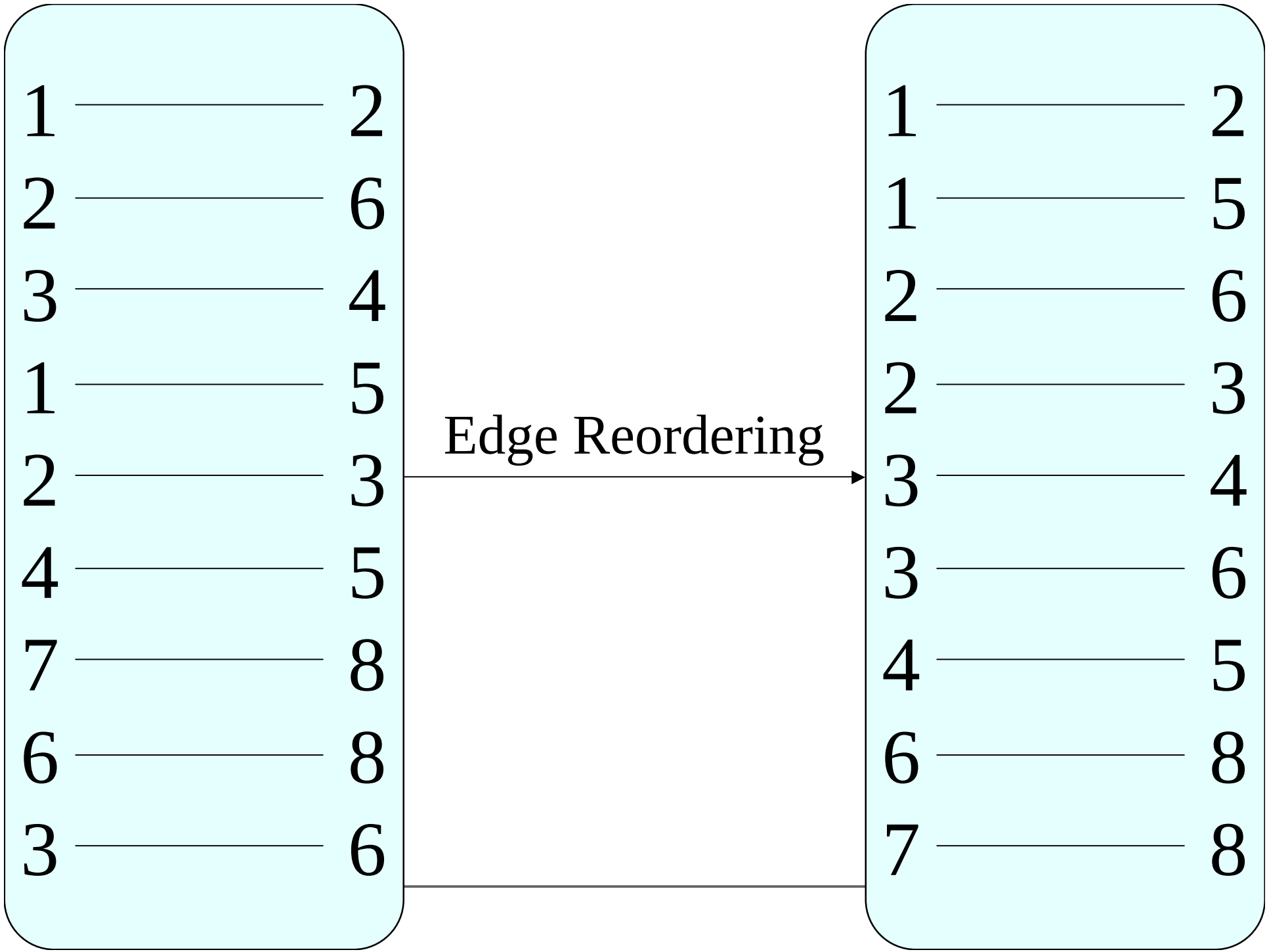
Hybrid Model: Implementation Issues

- Data Distribution
  - False sharing
  - Cache locality
- Work Division
  - Compiler or User
  - Static or dynamic
- Updates of the Shared Data
  - Private data but initialization and reductions are memory bandwidth bound
  - Shared data but updates need to be synchronized

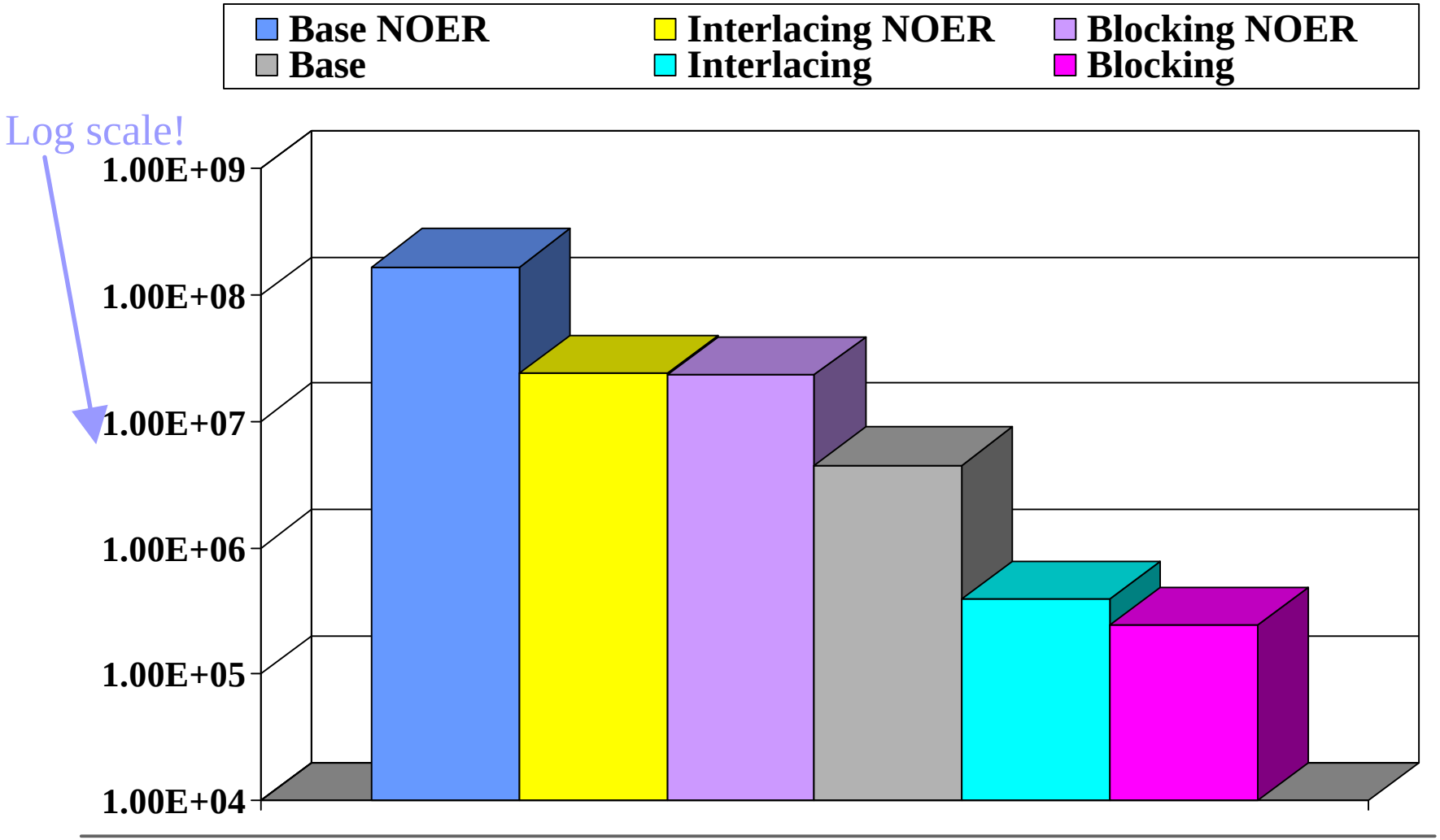
Hybrid Model: Three Implementation Strategies

- Edge Coloring
  - Poor cache locality
  - Compiler divides the work
  - Updates are independent
- Edge Reordering
  - Excellent cache locality
  - Compiler divides work
  - Updates are a problem
- Manual Work Division
  - Each MPI process calls MeTiS to further subdivide the work among threads
  - Boundary data is replicated for each thread
  - “Owner computes” rule is applied for every thread

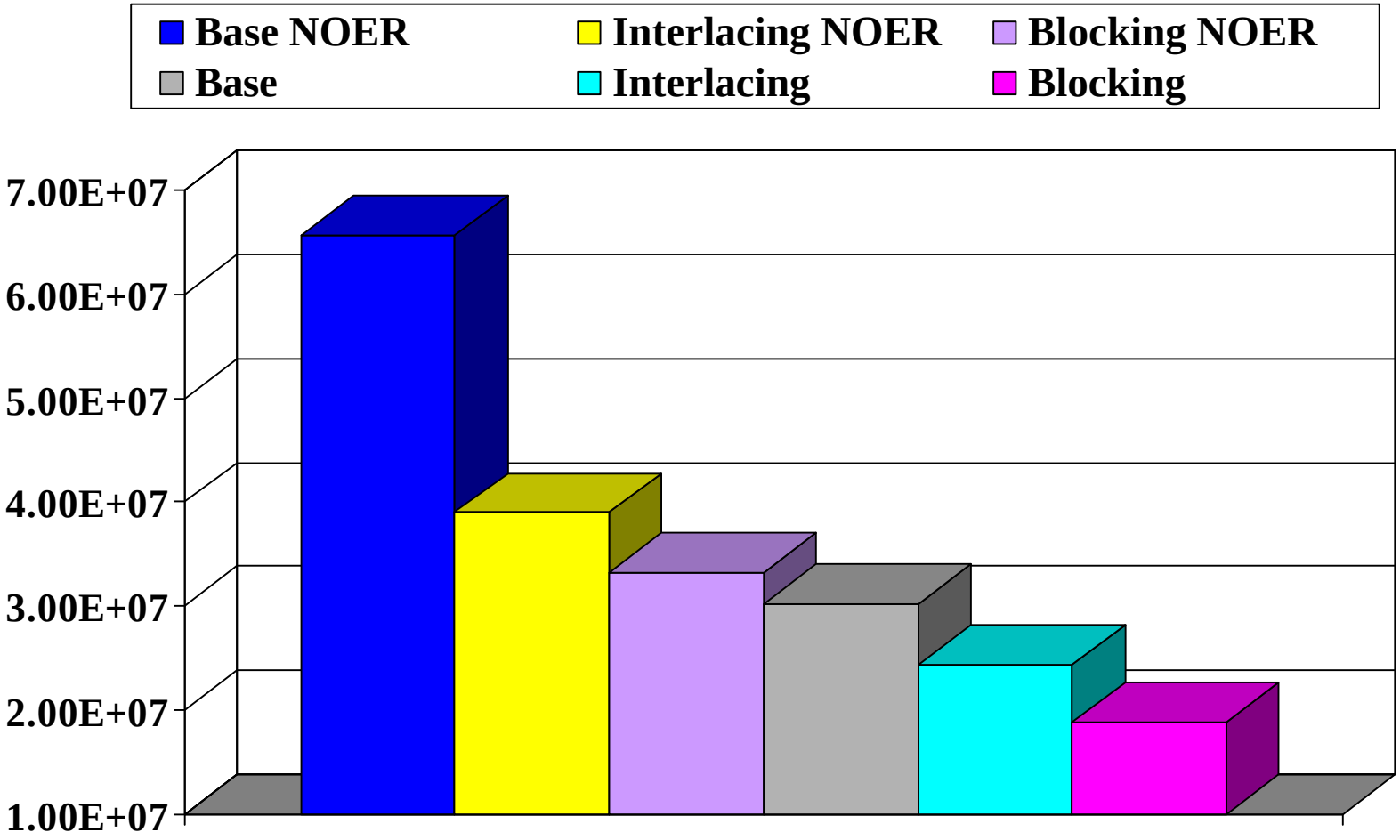

1 2
2 6
3 4
1 5
2 3
4 5
7 8
6 8
3 6
Edge Coloring
1 2
3 4
7 8
2 6
1 5
2 3
6 8
4 5
3 6


1 2
2 6
3 4
1 5
2 3
4 5
7 8
6 8
3 6
Edge Reordering
1 2
1 5
2 6
2 3
3 4
3 6
4 5
6 8
7 8

TLB Misses: Measured Values on Origin



Secondary Cache Misses: Measured Values on Origin



MPI/OpenMP in PETSc-FUN3D

- Only in the flux evaluation phase, as it is not memory bandwidth bound
- Gives the best execution time as the number of nodes increases because the subdomains are chunkier as compared to pure MPI case

Nodes	MPI Processes Per Node		MPI/OpenMP 2 Threads Per Node		
	1	2	Edge Coloring	Edge Reordering	MeTiS Divided
256	510	332	423	314	293
1024	183	136	130	116	109
3072	93	91	63	62	63

Conclusions

- OpenMP provides good support for incremental parallelism; however, needs attention to allow for software modularity, mixed language programming etc.
- Hybrid MPI/OpenMP achieves good overall performance but should be used only in the phases that are not memory bandwidth limited
  - Results in bigger subdomains
    - Faster convergence rate
    - Less network transactions

Acknowledgments

- Accelerated Strategic Computing Initiative, DOE
  - *access to ASCI Red and Blue machines*
- National Energy Research Scientific Computing Center (NERSC), DOE
  - *access to large T3E*
- SGI-Cray
  - *access to large T3E*
- National Science Foundation
  - *research sponsorship under Multidisciplinary Computing Challenges Program*
- U. S. Department of Education
  - *graduate fellowship support for D. Kaushik*

Related URLs

- Follow-up on this talk

  http://www.mcs.anl.gov/petsc-fun3d
- PETSc

  http://www.mcs.anl.gov/petsc
- FUN3D

  http://fmad-www.larc.nasa.gov/~wanderso/Fun
- ASCI platforms

  http://www.llnl.gov/asci/platforms
- International Conferences on Domain Decomposition Methods

  http://www.ddm.org
- International Conferences on Parallel CFD

  http://www.parcfd.org